Apple IIGS
#78: Bank Alignment and Memory Management

Revised by: Matt Deatherage May 1992
Written by: Matt Deatherage March 1990

This Technical Note discusses the way the Memory Manager deals with requests for memory that is already in use, and why this can be really annoying.
Changes since March 1990: Included new information about some smarter algorithms in System Software 6.0 and later which can avoid problems some of the time.

The Memory Manager is a sophisticated software module that provides the framework for the allocation, moving, management, and disposal of blocks of memory; however, it's not magic.

When you ask the Memory Manager for a block of memory and it's not immediately allocatable, the Memory Manager starts through the procedure for purging, compacting, and calling out-of-memory (OOM) queue routines until, at the end of its rope, it finally gives up and returns error $0201. The exact procedure is repeated below, taken from Volume 3 of the *Apple IIGS Toolbox Reference*. Note that each successive step is only taken if, after the previous step, the requested memory still isn't available.

1. Calls each OOM queue routine until either all routines have been called or until one OOM queue routine reports that it has freed enough memory to satisfy the request.
2. Compacts memory.
3. Purges all level 3 handles. If this frees enough memory, compaction occurs.
4. Purges all level 2 handles. If this frees enough memory, compaction occurs.
5. Purges all level 1 handles. If this frees enough memory, compaction occurs.
6. Calls each OOM queue routine again until all have been called or until one OOM queue routine reports that it has freed enough memory to satisfy the request.
7. Gives up and returns error $0201.

This strategy works pretty well—as long as the request is for a block of memory wherever it fits. If someone has asked the Memory Manager for memory at a specific address, things get stickier.

Suppose that you've asked the Memory Manager for a handle starting at the beginning of bank 2, and that something else (i.e., the ProDOS FST) is already using that memory. The Memory Manager notices that the handle isn't immediately available, so it starts going through the listed procedures. Since the handle for the ProDOS FST is neither purgeable nor movable and GS/OS isn't likely to give it up in an OOM queue routine, the request fails and the Memory Manager returns error $0201.

However, the Memory Manager went through **all** the steps listed to get to the seventh step, the error. The Memory Manager has no way to know that one of the OOM queue routines isn't going to give up that particular handle and allow the request to be fulfilled. The OOM queue routines

cannot know themselves, since they are only told how much memory is needed, not where it has to be. Therefore, whenever the Memory Manager returns error $0201, **all** purgeable handles have been purged.

This is particularly annoying to loaders. OMF supports a "bank-aligned" attribute for load segments, and the loaders ensure that such segments are loaded at the beginning of some bank or another. The Memory Manager does not have a "bank-aligned" attribute for handles, so the loaders have to do these things themselves. They do this by asking for a handle of the appropriate size at the beginning of bank two. If this fails, the loaders try again with bank three, then bank four, and so on through the end of memory.

Since some part of GS/OS is almost always occupying the memory at the beginning of bank two, which is where the loader first attempts to load a bank-aligned segment, the presence of such a segment in a load file virtually guarantees that all purgeable handles are purged when the file is loaded. This kicks out dormant applications and zombie-state tool sets, among other things, requiring they be loaded from disk again when needed.

Starting with System Software 6.0, the Loader attempts an alternate strategy first—it tries to allocate an entire bank of memory that is page aligned and doesn't cross a bank boundary. This block, if available, will by definition be bank-aligned. Since code segments can't be larger than 64K, such a block can always hold a bank-aligned segment. The Loader now tries to allocate such a block and if it finds one, it immediately disposes of it and allocates a block of the right size at the same bank address.

If this strategy succeeds, it's a lot faster than the other method and may avoid purging all of memory. If it fails, though, the Memory Manager still goes through all seven steps before returning error $0201, so all of memory may still be purged. It's just less likely in System Software 6.0 and later.

It doesn't make sense to bank-align a small segment, and small segments fit better into fragmented memory. If you use large segments anyway, consider the trade-off: bank-aligning a segment may purge memory at load time, but your linker may be able to generate smaller OMF, decreasing disk size and load time.

Summary

The general recommendation against asking for specific blocks of memory is well-known to most developers; the reasons outlined above simply add fuel to the fire against such programming practices. What isn't as widely known is that having a bank-aligned load segment in a load file may cause everything purgeable to be purged, and could also cause OOM queue routines to dispose of handles when there really isn't any kind of memory shortage.

Apple advises developers to carefully consider the advantages and disadvantages of bank-aligned segments before including one in a load file.

Further Reference

- *Apple IIGS Toolbox Reference*, Volumes 1 and 3
- *GS/OS Reference*